土木学会論文報告集
第339号・1983年11月
【論　　文】

# 構造解析のためのデータ構造とその適用例

# DATA STRUCTURE FOR STRUCTURAL ANALYSIS AND IT'S APPLICATIONS

結　城　皓　曠*
*By Teruhiro YUKI*

## 1. まえがき

近年，電子計算機による構造解析の手法は各種構造物の計画，新しい構造形式の開発および日常の設計業務に欠くことのできないものとなってきている．有限要素法による解析手法で，特に線形解析はその信頼性が広く認められるようになり，また数千元から一万元の連立一次方程式となるような大型問題の解析が，技術的にも経済的にも可能となり，複雑なモデルでできる限り実際の構造物に近い近似で解析が行われるようになっている．

しかし，このような大型で複雑なモデルを解析する場合，データの量は非常に大きくなり，解析のためのデータの作成および解析後のデータの評価が経済的にも技術的にも重要な課題となっている．このため，おのおのの工学分野で，いわゆるプリ・ポストプロセッサーの開発が要望され，着手されつつある．

構造解析の手法は，プログラム技術に関していえば，データをある大きさの配列で表示し，配列に対してある一定の手順で演算を実行するという比較的単純なプロセスが一般的であるが，大量の有限要素データを自動的に作成したり，解析結果に対して種々の後処理を行い，またこれらを図形表示し，会話形式で処理するプロセスの場合には，① 主記憶容量を越える多量のデータを取り扱う，② データの量が実行時でないと確定できない，③ 処理の手順が一定でない，④ データの階層が複雑で多重である，⑤ データの大きさや性質が多様である，などの特徴がある．したがって，プリ・ポストプロセスを含めて効率的な構造解析のシステムを実現し，しかも開発の効率を高めるためにはデータ構造を考慮することが有効であろう．

データ構造[1]の考え方は，オペレーションシステム等の計算機技術の分野に属し，構造解析のシステムに積極的に採用された例は少ない．ICES[2] (Integrated Civil Engineering System) の開発では初期の段階から考慮され，FORTRAN 言語の機能の制限をカバーした ICETRAN 言語を開発して，その中でダイナミックアレーを実現している．しかしこのシステムは莫大なもので，一般の計算機への移植性がなく，また通常の FORTRAN プログラムで使用することはできない．Bettess[3] は FORTRAN によってデータ構造を述べた少ない例の一つである．FORTRAN 使用者に示唆するところが多いが，ページングの手法は初歩的で効率がよいとはいえないし，データ構造については，一般性，拡張性に欠けるところがある．

著者は FORTRAN 言語のいくつかの制限をカバーでき，さらに構造解析の大型問題の処理に有効なデータ構造について考察した．またこのシステムを技術者の共通言語である FORTRAN を使って実現し，仮想配列の手法を開発してデータ構造を述べている．本論文は有限要素法における効果的なデータ構造の一手法としておもに次の項目について記している．

① 二次記憶領域を利用し，多量のデータを効率よく扱うためのページングの方法．

② 二次記憶領域を使用した仮想配列の方法，および連立一次方程式の解析への応用．

③ 仮想配列を利用した有限要素解析のためのデータ構造およびその適用例．

## 2. ページング

FORTRAN 言語の制約の一つは，主記憶領域（以下では主記憶という）に存在するデータに対してのみ演算処理が可能であり，したがって主記憶を超える多量のデータを扱うときはディスクなどの二次記憶領域（以下二次記憶という）にデータの大部分を記憶しておき，必要に応じて主記憶に転送して処理しなければならない．こ

* 正会員　石川島播磨重工業（株）技術研究所

の場合，主記憶のデータを二次記憶に転送して必要な領域を確保するなど主記憶の運用をプログラムで管理しなければならない．

この制約は，二次記憶の任意の番地の内容を自由に主記憶に転送する手法を用意すれば，あたかも主記憶のデータのごとく扱うことができ，取り除くことができる．

ページング[1]の方法は，主記憶内に定めたバッファー領域と二次記憶との間のデータ転送とバッファー領域の管理を自動的に行う方法で，次のように行われる．

① 二次記憶を一定の大きさのページ（PAGE）に分割して管理する．

② バッファーと二次記憶間のデータ転送はページ単位で行う．

③ バッファーは2ページ以上，場合によっては数十ページ分の大きさを用意する．

④ 二次記憶のある番地が参照されたとき，その番地を含むページ番号をバッファー上で探す．なければ不要なページを二次記憶に移し，その場所に所要のページを二次記憶から転送して，バッファー上の番地を明らかにする．

このような手順のプログラムを作成すれば，二次記憶の任意番地の内容を主記憶に転送（GET）したいとき，この方法で返されたバッファー内の番地にデータが転送されていることになり，ほとんど無限の大きさの二次記憶をあたかも主記憶のごとく扱うことができる．

以上の方法は，Bettess も基本的には同様な方法を述べており，また仮想記憶システムとしていくつかの計算機には基本ソフトウェアとして備えているものもあるが，ここでは，記憶容量の大小や機種によらず適用できるように基本的な FORTRAN 言語でまとめている．さらにデータ転送の効率向上のため次のような方法を採用してページングシステムを作成した．

① ページ番号の順にサイクリックにグループ番号を与え，バッファー内でページを探す際にはページ番号とグループ番号を参照することにしている．

② バッファー内のページ番号を記憶する方法は，グループごとに最近使用されたページと，より以前に使用されたページを区別しておき，後者は新しいものから順に参照できるようにしている．

③ バッファー内のページは内容が修正されたか否かを記録しておく．

④ あるページが参照されると，まずそのグループ番号で最近使用されたページ番号を調べ，次に以前に使用されたページ番号を順次新しいものから探す．所要のページ番号がみつからなければ，最も古いページを二次記憶に転送し（内容が修正されていなければ転送しないで）その場所に二次記憶から所要のページを転送して記憶する．

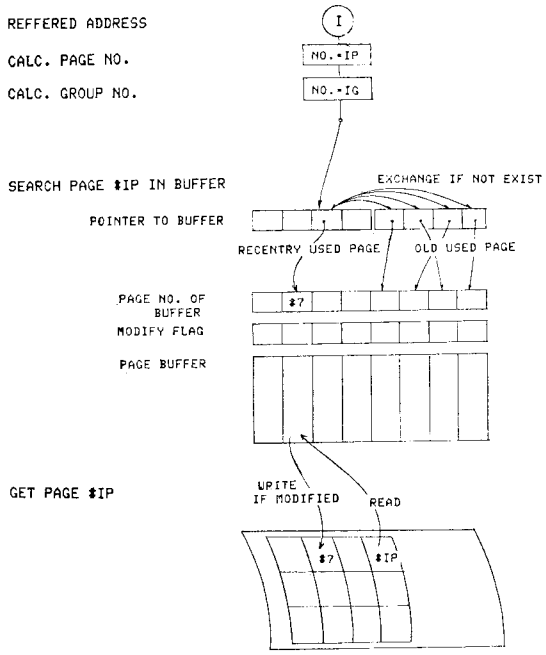



**Fig. 1** Paging System.

以上の方法はバッファー内に記憶しているページの中で最近使用されたページほど再び使用される確率が高く，古いものほど近い将来再使用される確率は低いという仮定に基づいている．これによって主記憶と二次記憶の間のデータ転送の回数を少なくしている．以上の方法の概要を **Fig. 1** に示している．FORTRAN プログラムは付録Ⅰに示している．

二次記憶の番地Ⅰの内容を参照するときは，

FUNCTION NADRS（I, IPUT）

を使用する．番地Ⅰを含むページがバッファー内に転送され NADRS にはバッファー内の番地がセットされ，この番地は主記憶領域であるから FORTRAN による各種の演算を施すことができる．たとえば二次記憶上の番地1の内容を 10 倍して番地3に記憶するときは

IB(NADRS(3, 1))＝IB(NADRS(1, 0))*10

である．ここで IB はバッファーである．

## 3. 仮想配列

FORTRAN の配列には次のような厳しい制限がある．

① 配列の大きさはコンパイルの時点で確定していなければならない．

② 配列は使用されるとされないにかかわらず全体が主記憶に存在する．

③ 多次元配列のサイズは方形でなければならない．

たとえば二次元配列では各行の要素の数は一定でなければならない．

このような制限のために，コンパイルの時点で予想される最大のサイズを定義しておく必要があり，実行時にはその大きさを超えることができない．

この制限をカバーするために次のような特長をもった仮想配列を考える．

① 主記憶でない仮想の領域に存在する．

② 実行時に定義できる．

③ 配列の一部を主記憶に GET したり，逆に主記憶のデータを仮想配列に転送 (PUT) したりすることが容易である．

④ 次の 4 種類の配列を使用できる．

a) 固定長一次元配列

b) 有限長一次元配列

c) 無限長一次元配列

d) 非方形二次元配列

このような仮想領域の配列は，先に述べたページングの方法を適用することによって実現できる．仮想領域は実際にはバッファーおよび二次記憶領域である．以下にその概要を述べ，その適用例として連立一次方程式の解法について記す．

### (1) 固定長一次元配列

この配列は，定義した時点で二次記憶の未使用域に所要の大きさの場所を確保し，その領域のスタート番地が配列名に与えられる．したがって定義後に配列の任意の要素を PUT または GET する場合はその配列名と配列上の位置を指示して行うことができる．配列は二次記憶に **Fig. 2** のように必要な大きさの場所が連続して確保される．FORTRAN プログラムは付録Ⅱの DEFVS, PUTVS および GETVS で，おのおの配列の定義，データの PUT, GET の機能をもっている．

この配列は，その大きさが確定している場合に適している．また，他の種類の仮想配列プログラムに使用される．

### (2) 有限長一次元配列

この配列は大きさがかなり大きくて不確定な一次元配列に適している．たとえば節点のデータはその数が解析する構造物によって 100 以下であったり数千を超える

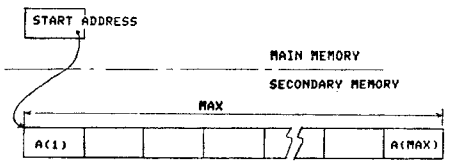



**Fig. 2** Virtual Array of Fixed Length.

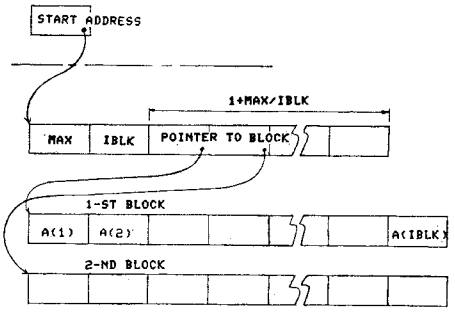



**Fig. 3** Virtual Array of Finite Length.

場合もあるが大きくとも 50 000 を超えないと予想される場合に使用すると有効である．この配列は最大の大きさを一括して確保するのではなく，ある大きさのブロックを単位として必要な部分だけを確保していく方法である．予想される最大の大きさとブロックの大きさを与えて定義するとこの時点では各ブロックの先頭番地を記憶するための場所のみが確保され，配列名にその場所の先頭番地がセットされる．配列の一部にデータが PUT されるとそのとき初めて所定のブロックが確保されデータが記憶されるようになっている．これらを図示すると **Fig. 3** となる．FORTRAN プログラムは付録Ⅱの DEFARY, PUTARY, GETARY および ADRARY である．前三者はそれぞれ配列の定義，データの PUT, GET, の機能をもっている．ADRARY は配列の任意の要素の二次記憶上での番地を求めるためのものであり，この配列は二次記憶上で連続して配置されないので必要な機能である．

### (3) 無限長一次元配列

配列の大きさを予想することが困難で，したがって実行中に拡張を必要とするような場合がある．このような場合には，必要なときに必要なだけ自動的に拡張していく配列が有効である．この配列は，最初に定義したときには，ある大きさのブロックを確保しておき，そのブロ

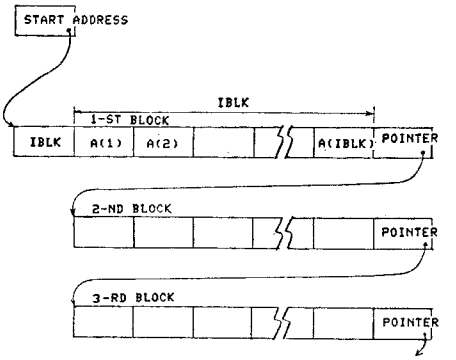



**Fig. 4** Virtual Array of Infinite Length.

ックを超えて使用したい場合には，自動的に第2ブロックを確保してその先頭番地を第1ブロックにポインターとして記憶するという方法で次々に拡張していく機能をもたせたものである．概要は **Fig. 4** に示している．FORTRAN プログラムは前記の固定長および有限長一次元配列の プログラム を使って 作成することができるが，付録 II に示すプログラム "DEFARY" の引数のうち，配列の長さを示す MSZ を 0 とすれば無限長配列が定義できる．

### （4） 非方形二次元配列

この配列は二次元配列の行ごとの要素数が異なり，しかも拡張が可能な機能をもった仮想領域の配列である．有限要素法のソリッド要素を構成する節点の数は4面体要素では4であるが中間節点をもった6面体要素では20というようにかなり差があり，このようなデータを一つの配列で表わすときには有効であろう．

**Fig. 5** に示すように，この配列は各行ごとに無限一次元配列を適用し，各行の先頭番地を有限一次元配列に記憶することによって実現できる．したがって前記の一次元仮想配列の プログラム を使って FORTRAN で容易に書くことができる．また，同様な考え方で三次元の非方形配列も可能である．

### （5） 連立一次方程式解析への適用

有限要素構造解析における連立一次方程式の解法については，これまで種々の直接解法[5]が提案されている．この中で，係数行列である構造全体の剛性行列が対称であること，また非零要素は対角線のまわりに分布するという 性質を利用し，この非零ゾーンについてのみ記憶し，演算するという方法[6]は最も効率的な方法の一つである．この方法は，係数行列（二次元配列）の各列の非零ゾーンの部分（アクティブカラム）のみを順に並べて一次元配列として扱うもので，そのために対角要素の一次元配列上の位置を記憶しておき，これを参照して二次元配列と一次元配列の位置関係を確保している．通常この一次元配列は主領域に記憶するが，方程式の元数が大きくなると二次記憶を使用しなければならない．このために，非零ゾーンの一次元配列をさらに適当な大きさのブロックに分割して記憶するとともに，主記憶の管理，ブロックの大きさを考慮した解析手順などをプログラム化する必要がある．

ここでは，一次元配列を二次記憶に直接記憶する代わりに，仮想一次元配列を適用して仮想領域に記憶（**Fig. 6**）し，各計算ステップで必要な部分のみ主記憶に取り出して計算を実行する．すなわち，係数行列を，

$$\boldsymbol{K} \Rightarrow \bar{\boldsymbol{K}}\boldsymbol{D}\bar{\boldsymbol{K}} \Rightarrow \boldsymbol{R}^T\boldsymbol{D}^{-1}\boldsymbol{R}$$

のように分解するとき，次のように列ごとに計算することができる．

$$\bar{K}_{ij} \Leftarrow K_{ij} - \sum_{m=1}^{i-1} R_{mi}\bar{K}_{mj} \quad (i=1, 2, \cdots, j-1)$$

$$\bar{K}_{jj} \Leftarrow K_{jj} - \sum_{m=1}^{j-1} D_{mm}\bar{K}_{mj}\bar{K}_{mj}$$

$$R_{ij} \Leftarrow D_{ii}\bar{K}_{ij} \quad (i=1, 2, \cdots, j-1)$$

$$D_{jj} \Leftarrow 1/\bar{K}_{jj}$$

この場合，変換すべき列 $j$ のほかに，必要な列を順次仮想配列から取り出して計算を行い，変換された列 $j$ を元の仮想配列に戻すという方法である．この方法によれば，主記憶と二次記憶の間のデータの転送はすべて仮想配列のシステムによって行われるので，連立方程式解析プログラムが簡単になるだけでなく，バンド幅や元数，主記憶容量等に関係なく必ず解を得ることができる．さらに有限要素データから構造全体の剛性行列を組み立てる場合にもページングの機能によって主記憶が管理されるので二次記憶との間のデータ移動が効率よく行われる．

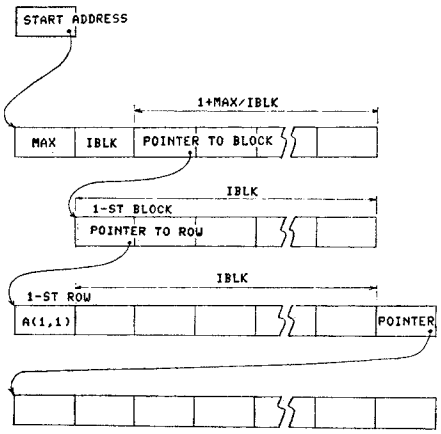



**Fig. 5** Virtual Array of 2-dimensional Non-sqared Shape.

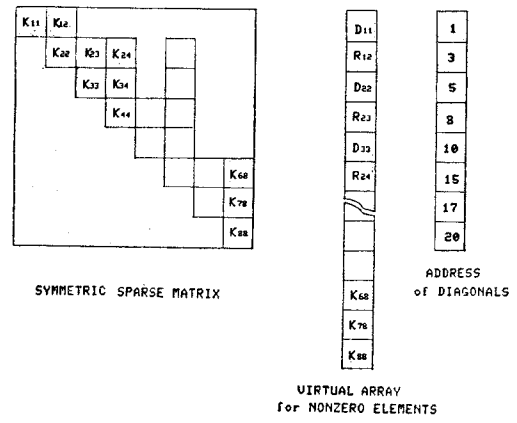
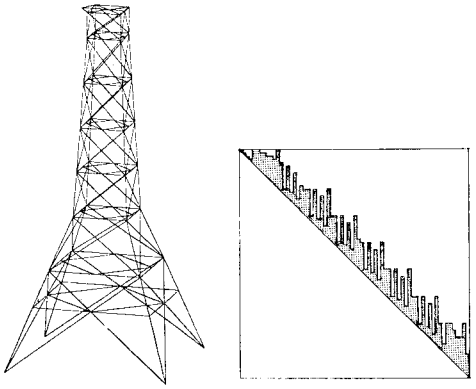



**Fig. 6** Stiffness Matrix in Virtual Array.



(a)　　(b)

**Fig. 7** Structure Model and Stiffness Matrix.

**Fig. 7** (a) の構造の連立方程式は自由度数 408，最大バンド幅 90，係数行列の非零ゾーンの形状は **Fig. 7**(b) のとおりでその大きさは 17 016 である．この問題をバッファーの大きさ 20 000 語のページングシステムで解析すると二次記憶を使わないで解析できたが，10 000 語（ページの大きさ 1 000 語，ページ数 10）として解析すると自動的にバッファーと二次記憶のデータ転送が行われ，READ, WRITE（主記憶から二次記憶へのデータの書き出しと読み込み処理）の回数は次のとおりであった．

剛性行列の組み立て：WRITE　8, READ　0
〃　三角化：WRITE 18, READ 18
求　解：WRITE 10, READ 26

このプログラムは一次記憶用に作成したものの一部を仮想配列用に書きかえることによって容易に作成することができる．

## 4. 有限要素データのデータ構造

有限要素解析において，構造モデルを表わす主要なデータは節点データ，要素データおよび要素と節点の親子関係を表わす要素節点データである．しかし，要素，節点は，構造物の物理的な特性をもったデータというより，むしろ解析の精度を確保するために構造部材を細分割した結果発生するデータである．一方，技術者にとっては，構造モデルを定義する場合，はり，平板，曲面板というように構造部材の物理的な特徴で分類して扱う方がなじみやすい．

一般に構造物は立体的な部分の固体 (Solid)，二次元的な広がりの面の部分

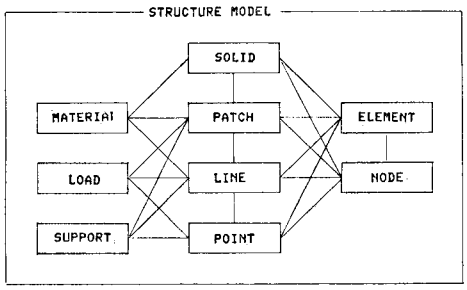


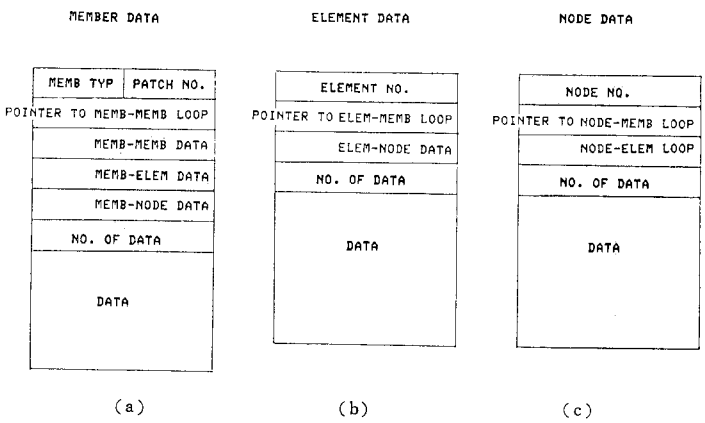

**Fig. 8** Data of Structure Model.

(Patch)，一次元的な線 (Line) および点 (Point) に分けることができる．これらの構成部分 (Member と呼ぶ) の間で，Solid は複数の Patch で構成され従属関係（親子関係）にあり，同様に Patch は複数の Line，Line は複数の Point というように親子関係の階層構造を形成している．また，Member は有限要素解析のために細分割されて要素 (Element) と節点 (Node) が作成されて階層構造を形成するので，これらの全体が構造モデルであると考えられる．**Fig. 8** は上に述べた幾何学的な Member のほかに材料データ (Material)，荷重データ (Load) および拘束データ (Support) を Member の一種として階層構造に加えて表示したものである．

このようにデータは Member, Element および Node の 3 種類に分類することができる．ただし Solid, Patch 等は Member の一種で，タイプが異なるものとして区別する．さらにこの 3 種のデータの相互の親子関係を表わすデータが 4 種類あり，次のように合計 7 種類のデータになる．

① Member データ

Member の固有のデータで，たとえば Patch の形状や曲面の種類，要素分割の方法等のデータである．その他に **Fig. 9**(a) のように Member の番号やタイプ番




(a)　　(b)　　(c)

**Fig. 9** Member, Element and Node data.

号，データの数および親子関係のデータへのポインターをもっている．ポインターは4つあり，子供の Member，Element および Node 番号を示す Member-Member データ，Member-Element データ，Member-Node データへのポインター，および親の Member を結ぶ Member-Member ループへのポインターである．

② Element データ

Member を細分割して生成した Element に関するデータで **Fig. 9**(b) に示すように，各 Element の形状や性質等の固有のデータのほかに，Element 番号，データ数および子供の Node 番号をもった Element-Node データへのポインター，親の Member 番号を結んだ Element-Member ループへのポインターをもっている．

③ Node データ

Member を細分割して生成した Node に関するデータで，**Fig. 9**(c) のように座標値等の固有データ，Node 番号，データ数およびポインターをもっている．ポインターは親の Element 番号を結んだ Node-Element ループおよび親の Member 番号を結んだ Node-Member ループへのポインターである．

④ Element-Node データ

Element を構成する子供の Node 番号をもったデータであるが **Fig. 10** (a) に示すように単に Node 番号だけをもつだけでなくて，親の Element 番号およびポインターの3つをセットとしてもっている．このポインターはすべての Element-Node データについて同じ番号の子供を次々と結んでループ（Node-Element ループ）を形成しておくと，このポインターをたどってループをひとまわりするとすべての親の Element 番号を効率よく調べることができる．

⑤ Member-Element データ

Member を構成する Element 番号をもったデータである．**Fig. 10**(b) のように Member 番号（およびタイプ番号）と Element 番号およびポインターの3つがセットになっている．ポインターはすべての Member-Element データの中で同じ子供の Element 番号を次々と結んでループ（Element-Member ループ）を形成するためのものである．このポインターをたどれば，すべての親の Member 番号を調べることができる．

⑥ Member-Node データ

Member を構成する Node 番号をもったデータである．**Fig. 10** (c) のように，Member-Element データと同様である．ポインターは Member-Node データについて，同じ Node 番号を結んで Node-Member ループを形成し，

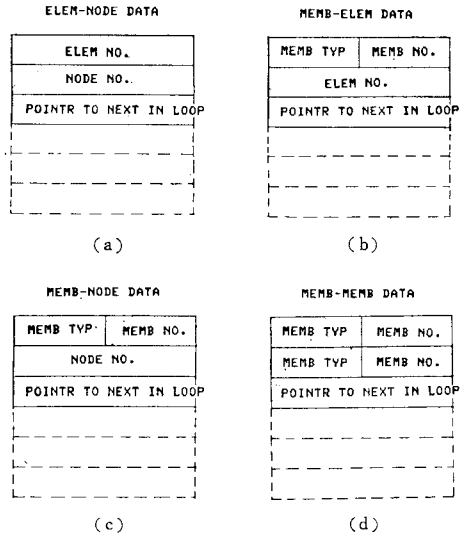



**Fig. 10** Relational Data.

これをたどることによって親の Member をすべて調べることができる．

⑦ Member-Member データ

Member を構成する子供の Member 番号をもったデータである．たとえば，Patch を構成する Line の番号あるいは Solid を構成する Patch の番号などである．**Fig. 10**(d) のように親の Member 番号と子供の Member 番号およびポインターをセットとしてもっている．ただし Member の場合は，番号と同時に種類を区別するためのタイプ番号をもつ必要がある．ポインターは親の Member を調べるための Member-Member ループを形成するものである．

以上のようなデータ構造は，データの階層構造を明確にし，特に子供のデータから親のデータを効率よく調べることができること，親子の関係を表わすデータは大き

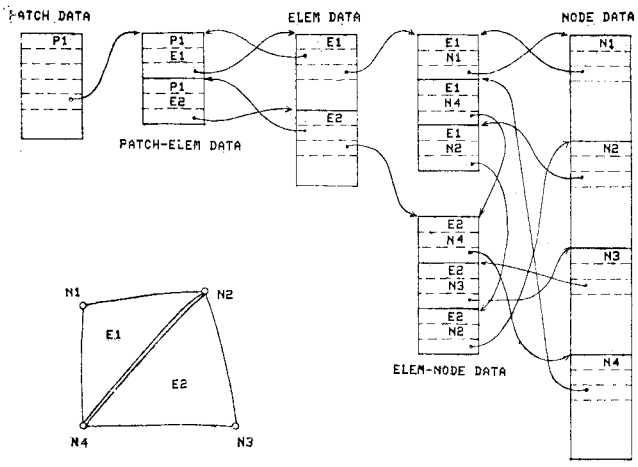



**Fig. 11** Example of Data and Relations.

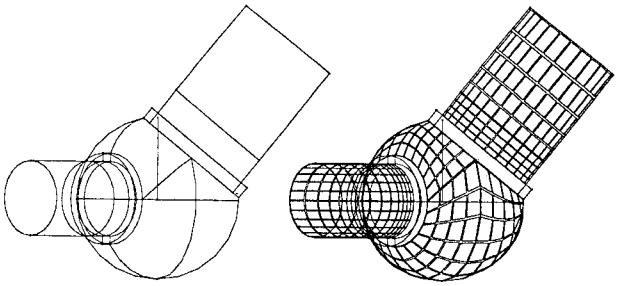



**Fig. 12** Example of Finite Element Model.

さが固定でなく自由にとれ，記憶場所も任意であるなどの特長がある．一方，単純な通常のデータ構造に比べて必要な記憶量は増え，ポインターの処理を自動的に行うシステムが必要になるが，前述の仮想配列を適用することによってこの問題は解決できる．すなわち，Element データに関していえば，各 Element ごとのデータ数は確定でき，Element 総数は実行中に定まるという二次元配列であるから，これには有限一次元配列を適用し，前三者には同様に扱う．次に Member-Node データは子供の Node 数が分割の方法によってまったく予想できないほど変化するので，おのおのの Member に対して無限長一次元配列を適用し，その先頭番地を Member データに記憶させる．後四者についてはこれと同様の扱いをする．また，ポインターは二次記憶の番地を使用する．このようにして，上記の7種のデータの定義，参照，修正等は，仮想配列のプログラムを使って FORTRAN で書くことができる．

**Fig. 11** は四辺形の Patch を2つの三辺形要素に分割したときのデータ構造の例を示している．Patch 1 Element 2, および Node 4 のデータおよび相互の親子関係を表わした例である．

**Fig. 12** は水圧鉄管の球分岐部の解析モデルである．**Fig. 12** (a) のように構造物を MEMBER (POINT, LINE, PATCH, SOLID の数はそれぞれ 29, 47, 29, 3) に分解して定義し，ELEMENT に自動分割した結果，NODE, ELEMENT がそれぞれ 662, 496 生成された．**Fig. 12**(b) は生成された板要素を図形表示したものである．このプログラムは対話形式で構造物の形状を定義し，三次元の図形を表示しながら自動分割によって ELEMENT, NODE を生成する機能をもっており，仮想配列を使った前記のデータ構造を適用している．**Fig. 12** のモデルの場合，データの大きさは約 55 000 語であったが，ページングのバッファー (10 240 語) の 5.5 倍のデータを扱っている．

## 5. まとめ

本報では，構造解析のためのデータ構造について考察し，その実用性について述べた．

データ量が多く，階層を構成する有限要素データを効率よく処理するために，まずページングの方法を開発し主記憶と二次記憶の間のデータ転送の自動化を図った．

次に，ページングの機能を使って仮想の記憶領域に定義できる仮想配列の方法を開発した．この仮想配列は実行中に定義でき，その大きさや形状に制限がなく，また使用中に拡張可能であるなど FORTRAN 言語の制限をカバーできる機能をもたせることができた．

構造解析におけるデータの階層を表現するために，データを NODE, ELEMENT, MEMBER の3種に分類し，相互間の親子関係を表わすデータ4種と合わせて7種に分類した．これらの各タイプのデータに仮想配列を適用した．

以上の機能を有限要素解析のプリプロセッサーに適用し，その有効性を確認した．

ここに述べたページング，仮想配列のプログラムはすべて FORTRAN で書いているので計算機の機種によらず一般に使用することができる．また，多量で階層構造をもつデータを扱う大型プログラムの開発が容易になり，工数削減に効果を発揮すると考える．

## 付録 I　ページングプログラム

プログラムの機能と引数の説明

① INITPG (KWORDS, IUSERS)
ページングシステムの初期設定を行う．
KWORDS：(IN), 二次記憶の大きさ (ページ数)
IUSERS：(OUT), ユーザーが使用可能な領域のアドレス

② RESTPG (KWORDS, IUSERS)
古いデータを使ってリスタートを行うときに使用する．

③ CLOSPG
一次記憶上のデータを2次記憶に PUT して終了する．

④ NADRS (II, IPUT)
二次記憶上のアドレスを指定すると，該当するページが一次記憶に読み込まれ，そのアドレスが NADRS にセットされる．
II：(IN), 二次記憶上のアドレス

```
C    ****************************************
C
C       PAGING SYSTEM
C
C          SUBROUTINE    INITPG  RESTPG  CLOSPG
C                        PGREAD  PGWRIT
C          FUNCTION      NADRS
C
C    ****************************************
C
      SUBROUTINE   INITPG(KWORDS,IUSERS)
C
C     ----- 1024 * IPAGSZ * NPAGE ------
C
      COMMON  /PAGING/NPAGE,IPAGSZ,MAXPG,NUNIT,ICPG,
     +                IOLDPG,LI,LS,IEXTNL,
     +                JPAGE,NGROUP,KGET,KPUT,KPAGE,IFILL,
     +                IGTPT(15),IPAGNO(15),LOC(15)
     +        /PAGE  /IBUF(10240)
      COMMON  /CMBULK/IBULK
      COMMON  /UNITNO/IRU,IWU
      DIMENSION  JBUF(1024,1)
      EQUIVALENCE  (IBUF(1),JBUF(1,1))
C
C     ------ NEW FILE ----------
      NEWOLD= 0
      GO TO 1
C
C     ------ RESTART --------------
      ENTRY   RESTPG(KWORDS,IUSERS)
C     ---------------------------------
      NEWOLD= 1
C
    1 NUNIT = 45
      MAXPG = KWORDS
      IF(MAXPG.LE.0) MAXPG = 300
      KPAGE=MAXPG
      NGROUP= 2**3
      NPAGE = 10
      JPAGE = NGROUP+1
      IPAGSZ= 2**10
C
      DEFINE  FILE  NUNIT(MAXPG,IPAGSZ,U,INX)
C
      IEXTNL= IPAGSZ*NPAGE
      ICPG  = NPAGE
      IOLDPG= 1
      LI    = 1
      LS    = 1
      KGET  = 0
      KPUT  = 0
C
      DO 10  I=1,NPAGE
      IPAGNO(I)= I
      LOC(I)   = I
      IGTPT(I)= 1
   10 CONTINUE
C
      IF(NEWOLD.NE.0) GO TO 50
C
C     ------- NEW FILE ------
      CALL   RCLEAR(IBUF,IEXTNL)
      IBULK = 3
      IUSERS= 3
C
      GOTO 1000
C
C     ------ OLD FILE -------
   50 DO 60 I=1,NPAGE
      READ(NUNIT'I) (JBUF(K,I),K=1,IPAGSZ)
   60 CONTINUE
      ICPG  = IBUF(1)
      IBULK = IBUF(2)
      IUSERS= 3
      GOTO 1000
C
C     ----- FILE CLOSE -----
      ENTRY   CLOSPG
C     ---------------------------------
      CALL  PUTUS(1,1,ICPG)
      CALL  PUTUS(2,1,IBULK)
      DO 110 I=1,NPAGE
      IF(IGTPT(I).EQ.0) GO TO 110
      IPAGE = IPAGNO(I)
      CALL   PGWRIT(IPAGE,JBUF(1,I))
  110 CONTINUE
 1000 RETURN
      END
C     ****************************************
C
      FUNCTION   NADRS(II,IPUT)
C
      COMMON      /UNITNO/IRU,IWU
C
      COMMON  /PAGING/NPAGE,IPAGSZ,MAXPG,NUNIT,ICPG,
     +                IOLDPG,LI,LS,IEXTNL,
     +                JPAGE,NGROUP,KGET,KPUT,IFILL(2),
     +                IGTPT(15),IPAGNO(15),LOC(15)
     +        /PAGE  /IB(1)
C
      NADRS = II
      IF(NADRS.LE.IEXTNL) RETURN
C
      I = II -1
      IADRS = MOD(I,IPAGSZ) +1
      IPAGE = I/IPAGSZ +1
      IF(IPAGE-IOLDPG) 5,100,5
    5 IF(IPAGE-MAXPG) 10,10,900
   10 CONTINUE
      IGROUP= MOD(IPAGE-1,NGROUP) +1
      LI    = LOC(IGROUP)
      IF(IPAGE-IPAGNO(LI)) 20,60,20
C
   20 IF(JPAGE.GT.NPAGE) GOTO 30
      DO 25  J=JPAGE,NPAGE
      LOC(IGROUP)= LOC(J)
      LOC(J)     = LI
      LI         = LOC(IGROUP)
      IF(IPAGE-IPAGNO(LI)) 25,60,25
   25 CONTINUE
C
   30 LS    = 1+(LI-1)*IPAGSZ
      IF(IGTPT(LI)) 40,50,40
C
   40 CALL   PGWRIT(IPAGNO(LI),IB(LS))
C
   50 IPAGNO(LI)= IPAGE
      IGTPT(LI)= IPUT
      NADRS= IADRS +LS -1
      IEXTNL= 0
      IOLDPG= IPAGE
      CALL   PGREAD(IPAGE,IB(LS))
      GOTO 1000
C
   60 CONTINUE
      LS    = 1+(LI-1)*IPAGSZ
      IOLDPG = IPAGE
C
C
  100 NADRS= IADRS +LS -1
      IF(IPUT.NE.0) IGTPT(LI)=1
C
      GOTO 1000
C
  900 WRITE(IWU, 901)
  901 FORMAT(1X,'---ERROR IN NADRS---SIZE OVER---')
 1000 RETURN
      END
C     ****************************************
C
      SUBROUTINE   PGREAD(IPAGE,IB)
      COMMON  /PAGING/NPAGE,IPAGSZ,MAXPG,NUNIT,ICPG,
     &               IFILL(6),KGET,KPUT
      DIMENSION    IB(IPAGSZ)
      DATA  IZERO/0/
C
      IF(IPAGE-ICPG) 11,11,10
   10 CALL RCLEAR(IB,IPAGSZ)
      GOTO 1000
   11 CONTINUE
      READ(NUNIT'IPAGE) IB
      KGET  = KGET +1
      GOTO 1000
C
      ENTRY         PGWRIT(IPAGE,IB)
      WRITE(NUNIT'IPAGE) IB
      KPUT  = KPUT +1
      IF(IPAGE-ICPG-1) 21,20,22
   20 ICPG  = IPAGE
   21 GOTO 1000
   22 ICPG  = ICPG +1
      IF(ICPG.EQ.IPAGE) GO TO 25
      WRITE(NUNIT'ICPG) (IZERO,I=1,IPAGSZ)
      GO TO 22
   25 CONTINUE
 1000 RETURN
      END
C     ************************************************
      SUBROUTINE  RCLEAR(K,KK)
C
      DIMENSION  K(1)
C
      DO 10 I=1,KK
      K(I)=0
   10 CONTINUE
      RETURN
      END
C     ****************************************
```

IPUT：(IN)，二次記憶へ PUT のとき=1
二次記憶から GET のとき=0

⑤ PGREAD (IPAGE, IB)
1ページ分のデータを二次記憶から主記憶へ GET する．
IPAGE：(IN)，ページ番号
IB：(IN)，データ

⑥ PGWRIT (IPAGE, IB)
1ページ分のデータを主記憶から二次記憶へ PUT する．
IPAGE：(IN)，ページ番号
IB：(OUT)，データ

COMMON 変数の説明
PAGING/
NPAGE：主記憶の大きさ（=10 ページ）
IPAGSZ：ページの大きさ（=1 024 語）
MAXPG：二次記憶の大きさ（ページ数）
NUNIT：二次記憶装置のユニット番号
ICPG：使用しているページ数
IOLDPG：直前にアクセスしたページの番号
LI：グループの占める場所へのポインター
LS：グループの占める場所のスタートアドレス
IEXTNL：データ量が NPAGE を超えると0となる．
JPAGE：NGROUP+1
NGROUP：グループの数（=8）
KGET：二次記憶から主記憶へ GET した回数
KPUT：主記憶から二次記憶へ PUT した回数
KPAGE, IFILL：不使用
IGTPT：主記憶上の各ページが修正されたかどうかを示す Flag
IPAGNO：主記憶上の各ページの番号
LOC：主記憶上の各ページのアドレス
/PAGE/
IBUF：ページを記憶する配列でその大きさは NPAGE *IPAGSZ(=10 240)
/CMBULK/
IBULK：二次記憶上の未使用領域のアドレス

## 付録 II 仮想配列プログラム

プログラムの機能と引数の説明
① DEFVS (NAME, MSZ)
固定長仮想配列を定義する．
NAME：(OUT)，仮想配列の二次記憶上のスタートアドレス
MSZ：(IN)，配列の大きさ

② GETVS (NAME, N, IA)
固定長仮想配列からデータを GET する．
NAME：(IN)，仮想配列の二次記憶上のスタートアドレス
N：(IN)，データの数
IA：(OUT)，データを記憶する配列

③ PUTVS (NAME, N, IA)
固定長仮想配列にデータを PUT する．
NAME：(IN)，仮想配列の二次記憶上のスタートアドレス
N：(IN)，データの数
IA：(IN)，データを記憶した配列

④ DEFARY (NAME, MSZ, ISZ)
有限長および無限長仮想配列を定義する．
NAME：(OUT)，仮想配列の二次記憶上のスタートアドレス
MSZ：(IN)，仮想配列の大きさ
有限長のとき ≒0，無限長のとき =0 とする．
ISZ：(IN)，仮想配列のブロックの大きさ

⑤ ADRARY (NAME, I, KADR)
仮想配列上の位置を指定し，二次起記上のアドレスを求める．
NAME：(IN)，仮想配列の二次記憶上のスタートアドレス
I：(IN)，仮想配列上の位置
KADR：(OUT)，二次記憶上のアドレス

⑥ GETARY (NAME, I, N, IDATA)
仮想配列上のデータを GET する．
NAME：(IN)，仮想配列の二次記憶上のスタートアドレス
I：(IN)，配列上の位置
N：(IN)，データの個数
IDATA：(IN)，データの入っている配列

⑦ PUTARY (NAME, I, N, IDATA)
データを仮想配列に PUT する．
NAME, I, N は GETARY の場合と同じ
IDATA：(OUT)，データを記憶する配列

COMMON 変数の説明
/CMARAY/
NAMOLD：直前にアクセスした仮想配列のアドレス
MAXOLD：直前にアクセスした仮想配列のサイズ
IBLOLD：直前にアクセスした仮想配列の各ブロックのサイズ

```
C     ******************************************
C
C        VIRTUAL ARRAY SYSTEM
C
C          SUBROUTINE  DEFVS   GETVS   PUTVS
C                      DEFARY  ADRARY  GETARY  PUTARY
C
C     ******************************************
      SUBROUTINE  DEFVS(NAME,MSZ)
C
      COMMON  /UNITNO/IRU,IWU
      COMMON  /CMBULK/IBULK
      COMMON  /PAGING/NPAGE,IPAGSZ,I000(58)
     +       /PAGE  /IB(1)
      DIMENSION   IA(1)
C
      NAME = IBULK
      IBULK = IBULK +MSZ
      GOTO 1000
C
      ENTRY         GETVS(NAME,N,IA)
C     ******************************
C
      IGTPT=1
      IF(NAME.LE.0) GO TO 900
      IF(N-1) 12,11,25
11    J     = NADRS(NAME,0)
      IA(1) = IB(J)
12    GOTO 1000
C
      ENTRY      PUTVS(NAME,N,IA)
C     ******************************
C
      IGTPT=2
      IF(NAME.LE.0) GO TO 910
      IF(N-1) 22,21,25
21    J     = NADRS(NAME,1)
      IB(J) = IA(1)
22    GOTO 1000
C
25    J     = NADRS(NAME,IGTPT-1)
      NN    = IPAGSZ - MOD(NAME-1,IPAGSZ)
      NEND  = MIN0(N,NN)
      NSTT=1
      GOTO 201
C
200   II= NAME+NEND
      NN    = NEND+IPAGSZ
      NSTT  = NEND+1
      NEND  = MIN0(N,NN)
      J     = NADRS(II,IGTPT-1)
C
201.  CONTINUE
      GO TO (210,220),IGTPT
210   DO 215 K=NSTT,NEND
      IA(K)=IB(J)
      J=J+1
215   CONTINUE
      IF(N.EQ.NEND) GOTO 1000
      GOTO 200
C
220   DO 225 K=NSTT,NEND
      IB(J)=IA(K)
      J=J+1
225   CONTINUE
      IF(N.EQ.NEND) GOTO 1000
      GO TO 200
C
900   WRITE(IWU,901)
901   FORMAT(1X,'---ERROR IN GETVS---')
      GOTO 1000
C
910   WRITE(IWU,911)
911   FORMAT(1X,'---ERROR IN PUTVS---')
1000  RETURN
      END
C     ******************************************
C
      SUBROUTINE  DEFARY(NAME,MSZ,ISZ)
C
      COMMON  /CMBULK/IBULK
      COMMON  /CMARAY/NAMOLD,MAXOLD,IBLOLD,LOLD,LA00
      COMMON      /UNITNO/IRU,IWU
      DIMENSION  IDATA(1),KSZ(2)
      EQUIVALENCE  (KSZ(1),MAXSZ),(KSZ(2),IBLKSZ)
C
      IF(ISZ.LE.0) GO TO 8
      IF(MSZ) 1,1,2
1     CALL  DEFVS(NAME,ISZ+3)
      CALL  PUTVS(NAME,1,MSZ)
      CALL  PUTVS(NAME+1,1,ISZ)
      GOTO 1000
C
2     NN    = 1 +(MSZ-1)*ISZ
      CALL  DEFVS(NAME,NN+2)
      NAMOLD= NAME
      MAXOLD= MSZ
      IBLOLD= ISZ
      LOLD=0
      LA00=0
      CALL  PUTVS(NAME,2,MAXOLD)
      GOTO 1000
8     WRITE(IWU,9)
9     FORMAT(1X,'---ERROR IN DEFARY---SIZE=0---')
      GOTO 1000
C
      ENTRY   ADRARY(NAME,I,KADR)
C     ***************************
      IPUT  = 10
      III   = I
      GO TO 100
      ENTRY      GETARY(NAME,I,N,IDATA)
C     ***************************
      IPUT  = 0
      III   = I+N-1
      GO TO 100
C
      ENTRY      PUTARY(NAME,I,N,IDATA)
C     ***************************
      IPUT  =1
      III   = I+N-1
C
100   CONTINUE
      IF(I.LE.0) GO TO 91
      IF(NAME.EQ.NAMOLD) GO TO 19
      CALL  GETVS(NAME,2,MAXSZ)
      IF(IBLKSZ.LE.0) GO TO 95
      IF(MAXSZ) 10,10,20
C
10    JBLKSZ= IBLKSZ+1
      LA0   = NAME+2
      IE    = IBLKSZ
      LA1=MOD(I-1,IBLKSZ)
      NN =MIN0(N,IBLKSZ-LA1)
11    IF(I.LE.IE) GO TO 25
      L     = LA0+IBLKSZ
      CALL  GETVS(L,1,LA0)
      IF(LA0.GT.0) GO TO 12
      CALL  DEFVS(LA0,JBLKSZ)
      CALL  PUTVS(L,1,LA0)
12    IE    = IE+IBLKSZ
      GO TO 11
C
19    MAXSZ =MAXOLD
      IBLKSZ=IBLOLD
20    CONTINUE
      JBLKSZ= IBLKSZ
      IF(III.GT.MAXSZ) GO TO 94
      L     = (I-1)/IBLKSZ+2+NAME
      LA1=MOD(I-1,IBLKSZ)
      NN =MIN0(N,IBLKSZ-LA1)
      IF(L.NE.LOLD) GO TO 24
      LA0=LA00
      GOTO 25
24    CALL  GETVS(L,1,LA0)
      IF(LA0.GT.0) GOTO 25
C
26    IF(IPUT.NE.0) GOTO 21
      CALL  RCLEAR(IDATA,NN)
      GOTO 28
C
21    CALL  DEFVS(LA0,IBLKSZ)
      CALL  PUTVS(L,1,LA0)
25    IADR  = LA0+LA1
C
      IF(IPUT.EQ.0) CALL  GETVS(IADR,NN,IDATA)
      IF(IPUT.EQ.1) CALL  PUTVS(IADR,NN,IDATA)
      IF(IPUT.NE.10) GOTO 28
      KADR=IADR
      GOTO 1000
C
28    NR    = N-NN
      IF(NR.LE.0) GOTO 50
29    I0    = NN +1
C
30    IF(MAXSZ.LE.0) L=LA0+IBLKSZ
      IF(MAXSZ.GT.0) L=L+1
      CALL  GETVS(L,1,LA0)
      IF(LA0.GT.0) GOTO 35
      IF(MAXSZ.LE.0) GOTO 31
      IF(IPUT.NE.0) GOTO 31
      NN=MIN0(NR,IBLKSZ)
      CALL  RCLEAR(IDATA(I0),NN)
      GOTO 38
31    CALL  DEFVS(LA0,JBLKSZ)
      CALL  PUTVS(L,1,LA0)
35    NN    = MIN0(NR,IBLKSZ)
      IF(IPUT.EQ.0) CALL  GETVS(LA0,NN,IDATA(I0))
      IF(IPUT.EQ.1) CALL  PUTVS(LA0,NN,IDATA(I0))
38    NR    = NR -NN
      IF(NR.LE.0) GOTO 50
40    I0    = I0 +NN
      GO TO 30
C
50    IF(MAXSZ.EQ.0) GOTO 90
      IF(LA0.EQ.0) GOTO 90
      NAMOLD=NAME
      MAXOLD=MAXSZ
      IBLOLD=IBLKSZ
      LOLD  =L
      LA00  =LA0
      GOTO 90
C
1000  CONTINUE
90    RETURN
C
91    WRITE(IWU,910) IPUT
      GO TO 90
94    WRITE(IWU,940) IPUT
      GO TO 90
95    WRITE(IWU,950) IPUT
      GO TO 90
C
910   FORMAT(1X,'---ERROR IN DEFARY------,IPUT=',I3)
940   FORMAT(1X,'---ERROR IN DEFARY---SIZE OVER---,IPUT=',I3)
950   FORMAT(1X,'---ERROR IN DEFARY---SIZE=0---,IPUT=',I3)
      END
```

LOLD：直前にアクセスした仮想配列のブロック番号

LA00：直前にアクセスした仮想配列のブロックのアドレス

## 参考文献


1) I. フローズ（久保寛彦訳）：データの構造，竹内書店，1972年8月．
2) Fenves, S.J., R.D. Logcher and S.P. March : STRESS; A Reference Manual, The M.I.T. Press, Jan. 1965.
3) Bettess, J.A. : A Data Structure for Finite Element Analysis, International Journal for Numerical Methods in Engineering, Vol. 11, No. 12, pp. 1779～1977, 1977年12月．
4) 山田 博：コンピュータ・アーキテクチャ，産業図書，1981年2月．
5) 成岡昌夫・中村恒善共編：骨組構造解析要覧，培風館，1976年4月．
6) Bathe, K. and E.L. Wilson : Numerical Methods in Finite Element Analysis, Prentice-Hall, Inc., 1976.


（1981.9.24・受付）